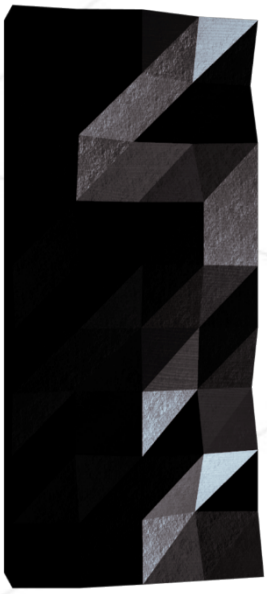

リブフト

あなただけのオリジナルライブラリーを

概要
このプロジェクトは、c言語のライブラリをコーディングするものです。
これは、あなたのプログラムが依存する汎用的な関数を多く含んでいます。

バージョン：15

目次

第1章 はじめに

C言語のプログラミングは、非常に便利な標準関数にアクセスできない場合、非常に退屈なものになります。このプロジェクトは、これらの関数がどのように機能するかを理解し、実装し、使いこなすことを目的としています。あなた自身のライブラリーを作成します。これは、次のC言語の学校の課題で使うことになるので、役に立つでしょう。

年間を通して、じっくりと自分のリブフトを拡張してください。ただし、新しいプロジェクトに取り組む際には、ライブラリで使用されている関数がプロジェクトのガイドラインで許可されているかどうかを確認することを忘れないでください。

第二章

共通事項

- プロジェクトはC言語で書かれている必要があります。
- あなたのプロジェクトは、ノルムに則って書かれていなければなりません。ボーナスファイル/関数がある場合、それらはノームチェックに含まれ、内部にノームエラーがある場合は0が表示されます。
- 未定義の動作とは別に、関数が予期せず終了する（セグメンテーションフォルト、バスエラー、ダブルフリーなど）ことがないようにしてください。このような場合、プロジェクトは非機能とみなされ、評価時に「0」となります。
- ヒープに割り当てられたすべてのメモリ空間は、必要なときに適切に解放されなければならない。リークは許されない。
- もし主題がそれを必要とするならば、あなたは、フラグ-Wall、-Wextra、-Werrorであなたのソースファイルを必要な出力にコンパイルするMakefileを提出し、ccを使用し、あなたのMakefileは再リンクしてはならないでしょう。
- Makefileには、少なくとも$(NAME)、all、clean、fcleanのルールが含まれていなければなりません。
  を
- あなたのプロジェクトにボーナスを回すには、Makefileにルールボーナスを含める必要があります。ボーナスは、プロジェクトのメイン部分で禁止されている様々なヘッダー、librairiesまたは関数をすべて追加します。ボーナスは、主題が何も指定しない場合は、別のファイル_bonus.{c/h}にする必要があります。必須部分とボーナス部分の評価は別々に行われます。
- プロジェクトがあなたの libft を使用することを許可する場合、そのソースと関連する Makefile を libft フォルダにコピーし、その関連する Makefile もコピーする必要があります。あなたのプロジェクトのMakefileは、そのMakefileを使ってライブラリをコンパイルしてから、プロジェクトをコンパイルしなければなりません。
- **提出の必要はなく、採点もされませんが**、プロジェクトのテストプログラムを作成することをお勧めします。自分の作品や仲間の作品を簡単にテストすることができます。このテストは、ディフェンスの時に特に役に立ちます。実際、ディフェンスでは、自分のテストや評価する仲間のテストを自由に使用することができます。
- 指定された git リポジトリに作品を提出すること。git リポジトリにある作品だ

けが採点されます。Deepthoughtがあなたの作品の採点を担当することになった場合、採点は以下のように行われます。

相互評価の後Deepthoughtの採点中にあなたの作品のいずれかのセクションでエラーが発生した場合、評価は停止されます。

第III章 必須パー

ト

プログラム名	libft.a
ファイルを提出する	Makefile、libft.h、ft_*.c。
メイクファイル	NAME、All、 Clean、Fclean、Re
外部ファンクション	以下、詳細
リブフト認定	n/a
商品説明	自分だけのライブラリを書こう　　関数の集合体 あなたのカーサスに役立つツールになることでしょう。

III.1　技術的な検討事項

- グローバル変数の宣言は禁止されています。
- より複雑な関数を分割するためにヘルパー関数が必要な場合は、静的関数として定義してください。こうすることで、その範囲は適切なファイルに限定されます。
- リポジトリのルートにすべてのファイルを配置します。
- 未使用のファイルの返却は禁止されています。
- すべての.cファイルは、-Wall -Wextra -Werrorのフラグでコンパイルする必要があります。
- ライブラリの作成には、必ずコマンドarを使用してください。libtoolコマンドを使用することは禁じられています。
- libft.a は、リポジトリのルートに作成する必要があります。

III.2　第1部】 Libcの機能

まず始めに、libcにある関数のセットを作り直す必要があります。あなたの関数は、オリジナルと同じプロトタイプを持ち、同じ動作を実装することになります。また、manで定義されている方法に従わなければなりません。唯一の違いは、その名前です。これらは、'ft_'というプレフィックスで始まります。例えば、strlenはft_strlenになります。

やり直さなければならない関数のプロトタイプの中には、「restrict」修飾子が使われているものがあります。このキーワードは、c99標準の一部です。したがって、このキーワードを自分のプロトタイプに含めることや、-std=c99フラ

以下のオリジナル関数を実装した独自の関数を作成する必要があります。これらは外部関数を必要としません：

- イサルファ
- イズディジット
- とうちゅう
- アイサシイ
- イズプリント
- ストレン
- ミームセット
- ビーゼロ
- ミームクパイ
- メモリムーブ
- ストラスクパイ
- ストラスカット

- おおとり
- トゥーロワー
- ストラー
- ストラーチ
- ストレングスエムピー
- メンヘル
- ミームクープ
- ストリングス
- アトイ

以下の2つの関数を実装するために、malloc()を使用することになります：

- キャロック
- ストラッドアップ

III.3　第2部】追加機能

この第2部では、libcにない機能、あるいはlibcに含まれているが別の形になっている機能のセットを開発する必要があります。

以下の関数の中には、Part 1の関数を書くのに便利なものがあります。

機能名	ft_substr
プロトタイプ	char *ft_substr(char const *s, unsigned int start、size_t len)です ；
ファイルを提出する	-
パラメータ	s:　s: 部分文字列を作成する元の文字列 start：　　文字列 's' の中の部分文字列の開始インデックス。 len:部分文字列の最大長を　　指定します。
戻り値	部分文字列のことです。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マリオック
商品説明	文字列 's' から部分文字列を (malloc(3) で) 割り当て、返す。 部分文字列は、インデックス 'start' から始まり、以下のようなものです。 最大サイズ 'len'。

機能名	ft_strjoin
プロトタイプ	char *ft_strjoin(char const *s1, char const *s2) ；
ファイルを提出する	-
パラメータ	s1:　プレフィックス文字列。 s2:　サフィックス文字列。
戻り値	新しい文字列です。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マリオック
商品説明	を連結した結果である新しい文字列を（ malloc(3) で ）確保し，返す。 の's1'と's2'です。

機能名	ft_strtrim
プロトタイプ	char *ft_strtrim(char const *s1, char const *set) ;
ファイルを提出する	-
パラメータ	s1: トリミングされる文字列。 のセットです： トリミングする文字の参照セット。
戻り値	切りそろえた系。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マリオック
商品説明	set' で指定された文字を削除した 's1' のコピーを (malloc(3) で) 割り当て、返す。 を、文字列の先頭と末尾から削除します。

機能名	ft_split
プロトタイプ	char **ft_split(char const *s, char c) ;
ファイルを提出する	-
パラメータ	s: 分割する文字列。 c: デリミタ文字。
戻り値	分割の結果得られた新しい文字列の配列。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マロック、フリー
商品説明	文字 'c' を区切り文字として 's' を分割して得られる文字列の配列を (malloc(3) で) 割り当て、返す。 配列の末尾は をNULLポインタで返します。

機能名	ft_itoaさん
プロトタイプ	char *ft_itoa(int n) ;
ファイルを提出する	-
パラメータ	n: 変換する整数を指定します。
戻り値	整数を表す文字列。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マリオック
商品説明	引数として受け取った整数を表す文字列を（ malloc(3) で ）確保し、返す。 負の数の取り扱いが必要です。

機能名	ft_strmapi
プロトタイプ	char *ft_strmapi(char const *s, char (*f) (unsigned) int, char))である ;
ファイルを提出する	-
パラメータ	s: イテレートする文字列。 f: 各キャラクタに適用する関数です。
戻り値	'f'を連続的に適用して作成される文字列。 割り当てに失敗した場合はNULLを返す。
外部ファンクション	マリオック
商品説明	文字列 's' の各文字に関数 'f' を適用し、そのインデックスを第一引数に渡して新しい文字列を作成する（ malloc(3) を使用 ）結果、次のようになります。 f'の連続的な適用によるものである。

機能名	ft_striteriさん
プロトタイプ	void ft_striteri(char *s, void (*f)(unsigned int、char* ））となります ;
ファイルを提出する	-
パラメータ	s: イテレートする文字列。 f: 各キャラクタに適用する関数です。
戻り値	なし
外部ファンクション	なし
商品説明	引数として渡された文字列の各文字に、そのインデックスを第一引数として、関数 'f' を適用する。 各文字は のアドレスを'f'に変更し、必要に応じて修正する。

機能名	ft_putchar_fd
プロトタイプ	void ft_putchar_fd(char c, int fd) ;
ファイルを提出する	-
パラメータ	c: 出力する文字。 fd： 書き込みの対象となるファイルディスクリプタ。
戻り値	なし
外部ファンクション	書く
商品説明	文字'c'を指定されたファイルに出力する。 ディスクリプタを使用します。

機能名	ft_putstr_fd
プロトタイプ	void ft_putstr_fd(char *s, int fd) ;
ファイルを提出する	-
パラメータ	s: 出力する文字列。 fd： 書き込みの対象となるファイルディスクリプタ。
戻り値	なし
外部ファンクション	書く
商品説明	文字列 's' を指定されたファイルに出力します。 ディスクリプタを使用します。

機能名	ft_putendl_fd
プロトタイプ	void ft_putendl_fd(char *s, int fd) ;
ファイルを提出する	-
パラメータ	s: 出力する文字列。 fd： 書き込みの対象となるファイルディスクリプタ。
戻り値	なし
外部ファンクション	書く
商品説明	与えられたファイルディスクリプタに文字列 's' を出力する。 の後に改行が続く。

機能名	ft_putnbr_fd
プロトタイプ	void ft_putnbr_fd(int n, int fd) ;
ファイルを提出する	-
パラメータ	n: 出力する整数を指定します。 fd： 書き込みの対象となるファイルディスクリプタ。
戻り値	なし
外部機能。	書く
商品説明	整数'n'を与えられたファイルに出力する。 ディスクリプタを使用します。

第IV章 ボーナ

スパート

必須パートをクリアしたら、迷わずこの追加パートに挑戦してください。成功すれば、ボーナスポイントがもらえます。

メモリや文字列を操作する関数はとても便利です。しかし、リストを操作することはもっと便利であることがすぐにわかるでしょう。

リストのノードを表現するために、次の構造を使用する必要があります。その宣言をlibft.h ファイルに追加してください：

```
typedef struct     s_list
{
  ボイド           *content ;
  struct s_list    *nextです ;
}                  t_list ;
```

t_list struct のメンバは次のとおりである：

- の内容です：ノードに含まれるデータ。
  void * あらゆる種類のデータを格納することができます。
- next：次のノードのアドレス、または次のノードが最後のノードである場合はNULL。

Makefileにmake bonusルールを追加して、libft.aにボーナス関数を追加してください。

ボーナスパートは、必須パートがPERFECTである場合にのみ評価されます。完璧とは、必須パートが統合的に行われ、誤動作することなく動作することを意味します。必須条件をすべてクリアしていない場合、ボーナスパートはまったく評価されません。

リストを簡単に利用するために、以下の機能を実装します。

機能名	ft_lstnew
プロトタイプ	t_list *ft_lstnew(void *content) ;
ファイルを提出する	-
パラメータ	の内容です： ノードを作成するためのコンテンツです。
戻り値	新ノード
外部ファンクション	マリオック
商品説明	新しいノードを（ malloc(3)で ）割り当てて返す。メンバ変数 'content' は ，パラメータ 'content' の値で初期化される ． また，変数 'next' は NULL に初期化される。

機能名	ft_lstadd_front
プロトタイプ	void ft_lstadd_front(t_list **lst, t_list *new) ;
ファイルを提出する	-
パラメータ	lst： リストの最初のリンクへのポインタのアドレス。 を新規に作成する： されるノードへのポインタのアドレス。 を追加した。
戻り値	なし
外部ファンクション	なし
商品説明	リストの先頭にノード 'new' を追加する。

機能名	ft_lstsize
プロトタイプ	int ft_lstsize(t_list *lst) ;
ファイルを提出する	-
パラメータ	lst： リストの先頭。
戻り値	リストの長さ
外部ファンクション	なし
商品説明	リスト内のノードの数をカウントします。

機能名	ft_lstlast
プロトタイプ	t_list *ft_lstlast(t_list *lst) ;
ファイルを提出する	-
パラメータ	lst： リストの先頭。
戻り値	リストの最後のノード
外部ファンクション	なし

商品説明	リストの最後のノードを返します。

機能名	ft_lstadd_back
プロトタイプ	void ft_lstadd_back(t_list **lst, t_list *new) ;
ファイルを提出する	-
パラメータ	lst： リストの最初のリンクへのポインタのアドレス。 を新規に作成する： されるノードへのポインタのアドレス。 を追加した。
戻り値	なし
外部ファンクション	なし
商品説明	リストの末尾にノード 'new' を追加する。

機能名	ft_lstdelone
プロトタイプ	void ft_lstdelone(t_list *lst, void (*del)(void) *));
ファイルを提出する	-
パラメータ	lstです： 解放するノード。 del： 削除するために使用する関数のアドレス 内容です。
戻り値	なし
外部ファンクション	自由自在
商品説明	ノードをパラメータとして与え，パラメータとして与えられた関数 'del' を用いて，ノードの内容のメモリを解放し，ノードを解放する． のメモリが解放される。 'next' は解放されてはならない。

機能名	ft_lstclear
プロトタイプ	void ft_lstclear(t_list **lst, void (*del)(void) *));
ファイルを提出する	-
パラメータ	lst： ノードへのポインタのアドレス。 del： 削除するために使用する関数のアドレス ノードの内容を表示します。
戻り値	なし
外部ファンクション	自由自在
商品説明	関数 'del' と free(3) を使って，与えられたノードとその後継のノードを削除して解放する． 最後に、リストへのポインタを NULLです。

機能名	ft_lstiter
プロトタイプ	void ft_lstiter(t_list *lst, void (*f)(void *)) ;
ファイルを提出する	-
パラメータ	lst： ノードへのポインタのアドレス。 f: を反復処理するために使用される関数のアドレス。 をリストアップしています。
戻り値	なし
外部ファンクション	なし
商品説明	リスト'lst'を反復処理し、関数を適用する。 'f'を各ノードのコンテンツにかける。

機能名	ft_lstmap
プロトタイプ	t_list *ft_lstmap(t_list *lst, void *(*f)(void *)、void (*del)(void *)) ;
ファイルを提出する	-
パラメータ	lst： ノードへのポインタのアドレス。 f: リストの反復処理に使用される関数のアドレスです。 del： 削除するために使用する関数のアドレス 必要であれば、ノードの内容を
戻り値	新しいリストです。 割り当てに失敗した場合はNULLとする。
外部ファンクション	マロック、フリー
商品説明	リスト 'lst' を反復処理し、各ノードの内容に関数 'f' を適用する。 関数'f'を連続して適用した結果、新しいリストを作成する。 del」関数は、次のような用途に使われます。 必要に応じて、ノードの内容を削除する。

第五章

提出物および相互評価

通常通り、Git リポジトリに課題を提出してください。あなたのリポジトリ内の作品だけが、ディフェンスで評価されます。ファイル名が正しいかどうか、遠慮なく再確認してください。

リポジトリのルートにすべてのファイルを配置します。